On the efficient construction of multislices from recurrences

Böhlen, M ; Kasperovics, R ; Gamper, J

Abstract: Recurrences are defined as sets of time instants associated with events and they are present in many application domains, including public transport schedules and personal calendars. Because of their large size, recurrences are rarely stored explicitly, but some form of compact representation is used. Multislices are a compact representation that is well suited for storage in relational databases. A multislice is a set of time slices where each slice employs a hierarchy of time granularities to compactly represent multiple recurrences. In this paper we investigate the construction of multislices from recurrences. We define the compression ratio of a multislice, show that different construction strategies produce multislices with different compression ratios, and prove that the construction of minimal multislices, i.e., multislices with a maximal compression ratio, is an NP-hard problem. We propose a scalable algorithm, termed LMerge, for the construction of multislices from recurrences. Experiments with real-world recurrences from public transport schedules confirm the scalability and usefulness of LMerge: the generated multislices are very close to minimal multislices, achieving an average compression ratio of approx. 99%. A comparison with a baseline algorithm that iteratively merges pairs of mergeable slices shows significant improvements of LMerge over the baseline approach.

DOI: https://doi.org/10.1007/978-3-642-13818-8_5

On the Efficient Construction of Multislices from Recurrences

Romans Kasperovics[1], Michael H. Böhlen[2], and Johann Gamper[1]

[1] Free University of Bozen-Bolzano, Dominikanerplatz 3, 39100 Bolzano, Italy
[2] University of Zürich, Binzmühlestrasse 14, 8050 Zürich, Switzerland

Abstract. Recurrences are defined as sets of time instants associated with events and they are present in many application domains, including public transport schedules and personal calendars. Because of their large size, recurrences are rarely stored explicitly, but some form of compact representation is used. Multislices are a compact representation that is well suited for storage in relational databases. A multislice is a set of time slices where each slice employs a hierarchy of time granularities to compactly represent multiple recurrences.

In this paper we investigate the construction of multislices from recurrences. We define the compression ratio of a multislice, show that different construction strategies produce multislices with different compression ratios, and prove that the construction of minimal multislices, i.e., multislices with a maximal compression ratio, is an NP-hard problem. We propose a scalable algorithm, termed `LMerge`, for the construction of multislices from recurrences. Experiments with real-world recurrences from public transport schedules confirm the scalability and usefulness of `LMerge`: the generated multislices are very close to minimal multislices, achieving an average compression ratio of approx. 99%. A comparison with a baseline algorithm that iteratively merges pairs of mergeable slices shows significant improvements of `LMerge` over the baseline approach.

1 Introduction

A recurrent event is the association of the same information with multiple time instants, e.g., the departure times of bus 10A from stop P. Domenicani in direction north-west. We call such a set of time instants the *recurrence* of an event. Recurrences might easily become very large. For example, in the city of Bozen-Bolzano with 15 bus routes, the buses are making around 1,000 trips a day, visiting up to 20 stops per trip. In a half year period the corresponding schedule contains approximately 7.5 million departure times. Due to this large size, recurrences are rarely stored explicitly in databases, rather some form of compact representation is used.

Multislices [1], defined as sets of *time slices*, are a compact representation formalism with a number of good properties: high compression for common real-world recurrences, scalable relational representation, and easy interpretation and processing. A time slice employs a *hierarchy* of time granularities

M. Gertz and B. Ludäscher (Eds.): SSDBM 2010, LNCS 6187, pp. 42–59, 2010.

to compress a recurrence that follows a regular pattern. For example, slice $\lambda_1 = (yea\{7\}, wee\{0\text{-}25\}, day\{0\text{-}4\}, hou\{7\}, min\{0,25,55\})$ represents minutes 0, 25, and 55 past 7 from Monday to Friday in the first 26 weeks in 2007. A multislice groups a set of time slices allowing to represent recurrences with more complex patterns.

In order to benefit from the multislice representation we first must construct multislices. In this paper we address the construction of multislices with a given hierarchy from recurrences that are represented explicitly as sets of time instants.

Example 1. Figure 1(a) shows a fragment of an explicit representation of a schedule as it is stored in an existing application. The fragment shows the recurrence P_{10A} of departures of bus no. 10A in direction north-west from "P. Domenicani" which is the first stop on route 10A in that direction. The recurrence contains 3250 departure times in the first 26 weeks in 2007. The attributes of relation BUSEXP are the identifier of a route, a direction, the identifier of a stop, and a departure time. Figure 1(c) shows a compact representation of P_{10A} as multislice

BUSEXP

rtid	dir	seq	stid	depti
10A	nw	1	P. Domenicani	2007-01-01 07:00
10A	nw	1	P. Domenicani	2007-01-01 07:25
10A	nw	1	P. Domenicani	2007-01-01 07:55
10A	nw	1	P. Domenicani	2007-01-01 08:25
10A	nw	1	P. Domenicani	2007-01-01 08:55
10A	nw	1	P. Domenicani	2007-01-01 09:25
10A	nw	1	P. Domenicani	2007-01-01 09:55
⋮	⋮	⋮	⋮	⋮
10A	nw	1	P. Domenicani	2007-07-01 18:55

(a)

MSL

mid	sid
10A	λ_1
10A	λ_2

SLI

sid	lev	gid	xid
λ_1	1	*yea*	649
λ_1	2	*wee*	650
λ_1	3	*day*	651
λ_1	4	*hou*	652
λ_1	5	*min*	653
λ_2	1	*yea*	649
λ_2	2	*wee*	650
λ_2	3	*day*	651
λ_2	4	*hou*	654
λ_2	5	*min*	655

SEL

xid	st	en
649	7	7
650	0	25
651	0	4
652	7	7
653	0	0
653	25	25
653	55	55
654	8	18
665	25	25
665	55	55

(b)

$$M_{10A} = \{\lambda_1 = (yea\{7\}, wee\{0\text{-}25\}, day\{0\text{-}4\}, hou\{7\}, min\{0,25,55\}),$$
$$\lambda_2 = (yea\{7\}, wee\{0\text{-}25\}, day\{0\text{-}4\}, hou\{8\text{-}18\}, min\{25,55\})\}$$

(c)

Fig. 1. Different representations of recurrence P_{10A}: (a) explicit relational representation, (b) as multislice M_{10A} in a relational representation, (c) as multislice in a symbolic notation

M_{10A} that consists of two time slices. Figure 1(b) shows the relational representation of M_{10A}. Relation MSL groups time slices into multislices. Relation SLI stores time slices as sets of tuples ordered by hierarchy levels, where each tuple refers to a time granularity and a set of integers. Relation SEL stores the sets of integers as sets of non-adjacent, non-overlapping intervals.

In this paper we show that a recurrence can be represented with various multislices of different size. Smaller multislices provide higher compression ratios. We

establish a two-step process for constructing multislices with a given hierarchy of time granularities from a given recurrence. In the first step we construct a singular multislice where each slice has the required hierarchy and corresponds to a single time instant in the recurrence. In the second step we minimize the singular multislice by merging slices until no further merging is possible. The order in which the merging is done impacts the size of the resulting multislice. We prove that the construction of a minimal multislice representation is an NP-hard problem. We propose an algorithm, called `LMerge`, that merges time slices in an order imposed by the levels of the hierarchy (`LMerge` stands for level-wise merge). `LMerge` runs in $O(d^2 n \log n)$ time, where d is the depth of the hierarchy and n is the size of the recurrence. We analyze the performance of `LMerge` by constructing the worst cases, running experiments on the real-world data, and comparing it with a straightforward baseline algorithm.

The main contributions of this paper can be summarized as follows:

- We propose a two-step bottom-up process for the construction of multislices, where first a singular multislice is constructed followed by an iterative merging of slices.
- We show that different merging strategies produce multislices with different compression ratio, and we prove that the construction of minimal multislices (with maximal compression ratio) is NP-hard.
- We provide `LMerge`, a scalable approximation algorithm for the construction of multislices.
- We show empirically that `LMerge` is scalable and produces multislices that are close to minimal multislices with an average compression ratio of 99%.

The rest of the paper is organized as follows. Section 2 introduces preliminary concepts. In Section 3 we prove that the construction of minimal multislices is NP-hard and we propose `LMerge` algorithm with an analytical evaluation in Section 4. Section 5 reports about an empirical evaluation using real-world recurrences from bus schedules. The paper concludes with related work, conclusions, and future work.

2 Preliminaries

2.1 Time Domain and Granularities

We assume a *time domain*, $\mathcal{A}$, as a set of time instants equipped with a total order $\leq$ and isomorphic to the integers. A time granularity is a partitioning of a subset of $\mathcal{A}$ into non-empty intervals of time instants, termed *granules*. Examples of time granularities are minutes (*min*), hours (*hou*), days (*day*), weeks (*wee*), months (*mth*), and years (*yea*). We assume a *bottom granularity*, $G_\perp$, such that each granule of $G_\perp$ contains exactly one time instant. In our running example minutes represent the bottom granularity, and we use the ISO 8601:2004 notation to denote time instants, e.g., 2007-02-12 07:15. Granularity *day*, for instance, divides the time domain into granules of 1440 minutes. The granules of each

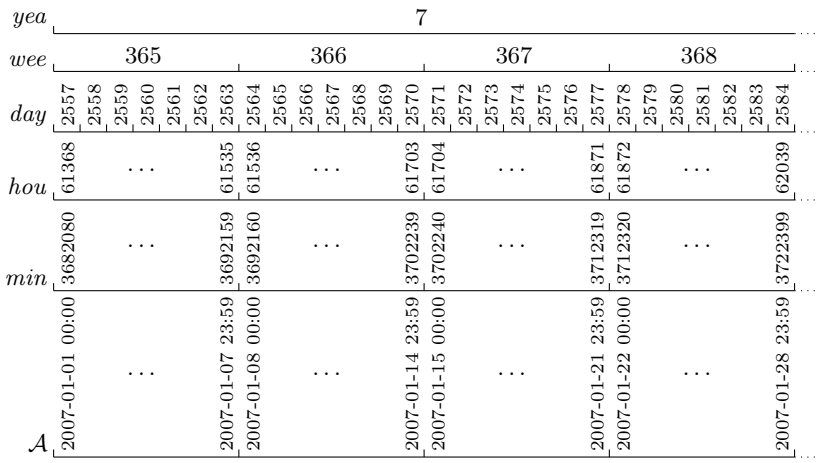


Fig. 2. Time domain and time granularities *min*, *hou*, *day*, *wee*, and *yea*

granularity G are ordered according to the time domain order and indexed with a subset of integers, $\mathcal{L}_G$, such that the indexing function $\mathcal{M}_G : \mathcal{L}_G \to G$ is an isomorphism that preserves the total order $\leq$. For each granularity we assume that the granule with index 0 contains time instant 2000-01-01 00:00. Figure 2 illustrates some correspondences of indexes between different granularities, e.g., $\mathcal{M}_{day}(2568) =$ [2007-01-12 00:00, 2007-01-12 23:59].

We adopt the *bigger-part-inside* [2,3] conversion between time granularities. The bigger-part-inside conversion of a granule $i \in \mathcal{L}_H$ of a granularity H to a granularity G, denoted $\lfloor^H_G(i)$, returns (the indexes of) those granules in G that are covered by granule i in H for more than a half or, if exactly half of a granule in G is covered, those with the second half covered, i.e.,

$$\lfloor^H_G(i) = \{j \mid (|\mathcal{M}_G(j) \cap \mathcal{M}_H(i)| > |\mathcal{M}_G(j) \setminus \mathcal{M}_H(i)|) \vee \\ (|\mathcal{M}_G(j) \cap \mathcal{M}_H(i)| = |\mathcal{M}_G(j) \setminus \mathcal{M}_H(i)| \wedge \max(\mathcal{M}_G(j)) \in \mathcal{M}_H(i))\}$$

2.2 Time Slices and Multislices

A (time) *slice* [4] is a finite list of pairs, $\lambda = (G_1X_1, \ldots, G_dX_d)$, where G_l are granularities and X_l are *selectors* that are defined as sets of integers. Each selector X_{l+1} specifies a set of granules in G_{l+1} with a relative positioning with respect to G_l. The sequence of granularities $(G_1, \ldots, G_d)$ is the *hierarchy* of a slice, and we assume that it always ends with the bottom granularity, i.e., $G_d = G_\perp$. Consider the slice $\lambda_1 =$ (*yea*{7}, *wee*{0-25}, *day*{0-4}, *hou*{7}, *min*{0,25,55}). The hierarchy is (*yea*, *wee*, *day*, *hou*, *min*). Selector {7} selects the year 2007, selector {0-25} selects the first 26 weeks in 2007, selector {0-4} selects the days from Monday to Friday from each of these weeks, etc.

The semantics of a slice $\lambda = (G_1X_1, \ldots, G_dX_d)$ is defined through the following mapping $\mathcal{I}$ to a subset of the time domain:

$$\mathcal{I}(\lambda) = \begin{cases} \bigcup_{k \in X_1} \mathcal{M}_{G_1}(k) & d = 1 \\ \mathcal{I}\left((G_2 \bigcup_{k \in X_1} (\lfloor^{G_1}_{G_2}(k)/X_2), \ldots, G_dX_d)\right) & d > 1 \end{cases}$$

A slice of depth $d = 1$ consists of a single granularity-selector pair and represents all time instants covered by those granules in G_1 selected by X_1. Otherwise, if $d > 1$, the slice is reduced to a slice of depth $d-1$ with hierarchy $(G_2, \ldots, G_d)$; $\lfloor^{G_1}_{G_2}(k)/X_2$ is defined as $\lfloor^{G_1}_{G_2}(k) \cap \{\min(\lfloor^{G_1}_{G_2}(k)) + i \mid i \in X_2\}$. Consider again the slice $\lambda_1 = (yea\{7\}, wee\{0\text{-}25\}, day\{0\text{-}4\}, hou\{7\}, min\{0,25,55\})$. First, years are mapped to weeks yielding $\mathcal{I}((wee\{365\text{-}390\}, day\{0\text{-}4\}, hou\{5\text{-}6\}, min\{15,35\}))$, then weeks are mapped to days, and so on, returning a total of 390 time instants.

A slice can be split into two slices by splitting one selector into two disjoint subsets [1]. The two slices represent disjoint sets of time instants, and their union is equal to the set represented by the original slice. For example, by splitting the selector of weeks the slice $(yea\{7\}, wee\{0\text{-}25\}, day\{0\text{-}4\}, hou\{7\}, min\{0,25,55\})$ can be split into $(yea\{7\}, wee\{0,2\text{-}25\}, day\{0\text{-}4\}, hou\{7\}, min\{0,25,55\})$ and $(yea\{7\}, wee\{1\}, day\{0\text{-}4\}, hou\{7\}, min\{0,25,55\})$.

A slice $\lambda' = (G_1X'_1, \ldots, G_dX'_d)$ is a *subslice* of a slice $\lambda = (G_1X_1, \ldots, G_dX_d)$, denoted $\lambda' \sqsubseteq \lambda$, if they both have the same hierarchy and $X'_l \subseteq X_l$ for all levels $l = 1, \ldots, d$. For example, the slice $(yea\{7\}, wee\{1\}, day\{0\text{-}4\}, hou\{7\}, min\{25,55\})$ is a subslice of $(yea\{7\}, wee\{0\text{-}25\}, day\{0\text{-}4\}, hou\{7\}, min\{0,25,55\})$. The subslice λ' represents a subset of the recurrence represented λ, i.e., $\lambda' \sqsubseteq \lambda \implies \mathcal{I}(\lambda') \subseteq \mathcal{I}(\lambda)$.

A *multislice* M is defined as a set of slices and represents all the time instants represented by the included slices, i.e., $\mathcal{I}(M) = \bigcup_{\lambda \in M} \mathcal{I}(\lambda)$. To simplify the operations on multislices, we require that all slices within a multislice have the same hierarchy. The *compression* of a multislice M is defined as $1 - \frac{|M|}{|\mathcal{I}(M)|}$. Referring to Example 1, recurrence P_{10A} with 3250 time instants is represented by multislice M_{10A} with two slices, which gives a compression ratio of $1 - \frac{2}{3250} = 99.94\%$.

3 Constructing Multislices from Recurrences

In this section we study the construction of multislices from recurrences. We adopt a bottom-up approach which first constructs a slice for each individual time instant in the recurrence and then iteratively merges these slices.

3.1 Basic Concepts

Definition 1 (Singluar Slice). *A slice* $\dot{\lambda} = (G_1X_1, \ldots, G_dX_d)$ *is* singular *if for all hierarchy levels* $l = 1, \ldots, d : |X_l| = 1$.

Singular slices have two important properties: (1) they represent exactly one time instant[1] and (2) two different singular slices cannot represent the same time instant. To facilitate reading, we put a dot over the slice symbol for singular slices, e.g., $\dot{\lambda}$.

If a slice represents a time instant it must have as a subslice a singular slice representing the same time instant.

Lemma 1. *For slices* $\lambda = (G_1X_1, \ldots, G_dX_d)$ *and* $\dot{\lambda} = (G_1\{x_1\}, \ldots, G_d\{x_d\})$, *if* $t \in \mathcal{I}(\lambda)$ *and* $t = \mathcal{I}(\dot{\lambda})$ *then* $\dot{\lambda} \sqsubseteq \lambda$.

We call a multislice that contains only singular slices a *singular multislice*. For a given hierarchy $(G_1, \ldots, G_d)$, each recurrence P can be represented by a unique singular multislice (provided that empty slices are excluded).

Example 2. Consider the input recurrence P = {2007-01-12 05:10, 2007-01-12 05:30, 2007-01-12 06:10, 2007-01-12 06:20, 2007-01-12 06:30, 2007-01-12 06:40, 2007-01-12 06:50, 2007-01-12 07:20, 2007-01-12 07:40, 2007-01-12 07:50, 2007-01-12 08:10, 2007-01-12 08:30} and the hierarchy (*yea*, *wee*, *day*, *hou*, *min*). The corresponding singular multislice is the following multislice M where each of the 12 time instants is represented with a distinct singular slice:

$$\begin{aligned}
M = \{&\dot{\lambda}_1 = (yea\{7\}, wee\{1\}, day\{4\}, hou\{5\}, min\{10\}),\\
&\dot{\lambda}_2 = (yea\{7\}, wee\{1\}, day\{4\}, hou\{5\}, min\{30\}),\\
&\dot{\lambda}_3 = (yea\{7\}, wee\{1\}, day\{4\}, hou\{6\}, min\{10\}),\\
&\dot{\lambda}_4 = (yea\{7\}, wee\{1\}, day\{4\}, hou\{6\}, min\{20\}),\\
&\dot{\lambda}_5 = (yea\{7\}, wee\{1\}, day\{4\}, hou\{6\}, min\{30\}),\\
&\dot{\lambda}_6 = (yea\{7\}, wee\{1\}, day\{4\}, hou\{6\}, min\{40\}),\\
&\dot{\lambda}_7 = (yea\{7\}, wee\{1\}, day\{4\}, hou\{6\}, min\{50\}),\\
&\dot{\lambda}_8 = (yea\{7\}, wee\{1\}, day\{4\}, hou\{7\}, min\{20\}),\\
&\dot{\lambda}_9 = (yea\{7\}, wee\{1\}, day\{4\}, hou\{7\}, min\{40\}),\\
&\dot{\lambda}_{10} = (yea\{7\}, wee\{1\}, day\{4\}, hou\{7\}, min\{50\}),\\
&\dot{\lambda}_{11} = (yea\{7\}, wee\{1\}, day\{4\}, hou\{8\}, min\{10\}),\\
&\dot{\lambda}_{12} = (yea\{7\}, wee\{1\}, day\{4\}, hou\{8\}, min\{30\})\}
\end{aligned}$$

Two slices $\lambda_X = (G_1X_1, \ldots, G_dX_d)$ and $\lambda_Y = (G_1Y_1, \ldots, G_dY_d)$ that have the same hierarchy can be merged into one slice of the same hierarchy iff at all hierarchy levels except one the corresponding selectors are equal, i.e., $X_m \neq Y_m$ for some level m and $X_l = Y_l$ for all levels $l \neq m$. We say also that λ_X and λ_Y are *mergeable* across level m, denoted as mergeable$(\lambda_X, \lambda_Y, m)$.

Definition 2 (Merge Operation). *Let* $\lambda_X = (G_1X_1, \ldots, G_dX_d)$ *and* $\lambda_Y = (G_1Y_1, \ldots, G_dY_d)$ *be two slices that are mergeable across level* m. *The* merge *operation of* λ_X *and* λ_Y *returns a slice and is defined as*

$$\lambda_X + \lambda_Y = (G_1X_1, \ldots, G_{m-1}X_{m-1}, G_mX_m \cup Y_m, G_{m+1}X_{m+1}, \ldots, G_dX_d)$$

[1] We assume that the selectors are consistent and exclude the cases when a slice represents the empty set due to inconsistent selectors.

Example 3. Consider the multislice M from the previous example. The slices $\dot{\lambda}_1$ and $\dot{\lambda}_2$ are mergeable across the level of *min*, since only at this level the corresponding selectors are different. The result of merging these two slices is ($yea\{7\}, wee\{1\}, day\{4\}, hou\{5\}, min\{10,30\}$). As another example, the slices $\dot{\lambda}_1$ and $\dot{\lambda}_4$ are not mergeable because the corresponding selectors for more than one granularity differ, namely *min* and *hou*.

The merge operation is *commutative*, i.e., for two mergeable slices λ_X and λ_Y, we have $\lambda_X + \lambda_Y = \lambda_Y + \lambda_X$, and it is *associative* only for slices that are mergeable across the same level, i.e., if mergeable$(\lambda_X, \lambda_Y, m)$ and mergeable$(\lambda_Y, \lambda_Z, m)$ then $(\lambda_X + \lambda_Y) + \lambda_Z = \lambda_X + (\lambda_Y + \lambda_Z)$.

3.2 Baseline Algorithm `BMerge`

Let $P \subset \mathcal{A}$ be a non-empty finite recurrence and $(G_1, \ldots, G_d)$ be a hierarchy with $G_d = G_\perp$. Our goal is to construct a multislice M with the given hierarchy such that $\mathcal{I}(M) = P$. Algorithm `BMerge` implements a baseline strategy for the bottom-up construction of a multislice and operates in two steps. First, for each time instant in the recurrence P a singular slice is constructed, yielding a singular multislice M that has the same size as the recurrence. The second step iterates over the slices in M. In each iteration a pair of mergeable slices is selected and merged into a single slice. The loop terminates when no more mergeable slices exist.

Algorithm `BMerge`

input: recurrence P, hierarchy $(G_1, \ldots, G_d)$
output: multislice M
// Step 1: build a singular multislice
$M := \varnothing$;
for each $t \in P$ **do**
  $M := M \cup \{\text{singular slice for } t\}$;
// Step 2: merge slices
while M contains mergeable slices **do**
  Select a pair $\lambda, \lambda' \in M$ such that mergeable(λ, λ', m);
  $M := M \setminus \{\lambda, \lambda'\} \cup \{\lambda + \lambda'\}$;
return M;

The result of `BMerge` is a *final* multislice, i.e., a multislice that contains no mergeable slices. Depending on the order in which pairs of mergeable slices are selected in Step 2 of the algorithm, different final multislices are obtained as shown in the following example.

Example 4. Consider the recurrence P from Example 2 and the corresponding singular multislice M which `BMerge` constructs in the first step. If in Step 2 the merging is done in the order indicated with parentheses $(\dot{\lambda}_1 + \dot{\lambda}_2) + (\dot{\lambda}_{11} + \dot{\lambda}_{12})$,

$(\dot{\lambda}_3 + \dot{\lambda}_4) + (\dot{\lambda}_5 + \dot{\lambda}_6)$, $(\dot{\lambda}_7 + \dot{\lambda}_{10})$, and $(\dot{\lambda}_8 + \dot{\lambda}_9)$, we get a final multislice

$$\begin{aligned} M_{\text{fin}} = \{&(\mathit{yea}\{7\}, \mathit{wee}\{1\}, \mathit{day}\{4\}, \mathit{hou}\{5{,}8\}, \mathit{min}\{10{,}30\}),\\ &(\mathit{yea}\{7\}, \mathit{wee}\{1\}, \mathit{day}\{4\}, \mathit{hou}\{6\}, \quad \mathit{min}\{10{,}20{,}30{,}40\}),\\ &(\mathit{yea}\{7\}, \mathit{wee}\{1\}, \mathit{day}\{4\}, \mathit{hou}\{6\text{-}7\}, \mathit{min}\{50\}),\\ &(\mathit{yea}\{7\}, \mathit{wee}\{1\}, \mathit{day}\{4\}, \mathit{hou}\{7\}, \quad \mathit{min}\{20{,}40\})\}. \end{aligned}$$

If, instead, we merge $((\dot{\lambda}_1 + \dot{\lambda}_2) + (\dot{\lambda}_3 + \dot{\lambda}_5)) + (\dot{\lambda}_{11} + \dot{\lambda}_{12})$ and $((\dot{\lambda}_4 + \dot{\lambda}_6) + \dot{\lambda}_7) + ((\dot{\lambda}_8 + \dot{\lambda}_9) + \dot{\lambda}_{10})$ we get a different final multislice

$$\begin{aligned} M_{\text{min}} = \{&(\mathit{yea}\{7\}, \mathit{wee}\{1\}, \mathit{day}\{4\}, \mathit{hou}\{5\text{-}6{,}8\}, \mathit{min}\{10{,}30\}),\\ &(\mathit{yea}\{7\}, \mathit{wee}\{1\}, \mathit{day}\{4\}, \mathit{hou}\{6\text{-}7\}, \quad \mathit{min}\{20{,}40{,}50\})\}. \end{aligned}$$

While both multislices are final, they have a different size and hence achieve a different compression ratio.

Definition 3 (Minimal Multislice). *Let P be a recurrence and $\mathsf{M} = \{M_1, \ldots, M_n\}$ be the set of all multislices with hierarchy $(G_1, \ldots, G_d)$ that represents P. A multislice $M_{min} \in \mathsf{M}$ is* minimal *iff $\forall M \in \mathsf{M}(|M| \geq |M_{min}|)$.*

A multislice representation with a given hierarchy of a recurrence P is minimal if all other multislices for P are of the same size or greater. A recurrence P can have more than one minimal multislice with a given hierarchy, where all have the same size. Minimal multislices provide the best compression ratio. A minimal multislice is always a final multislice, but not vice versa. In the above example, the final multislice M_{min} is also a minimal, which is not true for M_{fin}. Thus, for a recurrence P, a minimal multislice M_{min}, and a final multislice M_{fin} the following holds: $|M_{\text{min}}| \leq |M_{\text{fin}}| \leq |P|$.

The worst case complexity of `BMerge` is $O(d \cdot |P|^2)$. In this worst case after the first pass through the singular slices of M all mergeable slices are merged and appended at the "end" of the multislice. Further merging is only possible among these appended slices which result in new appended slices, etc.

3.3 NP-Hardness of Computing Minimal Multislices

In the following we show that searching for a minimal multislice representation is an NP-hard problem.

Definition 4 (Decomposition). *A singular multislice M is a* decomposition *of a slice $\lambda = (G_1 X_1, \ldots, G_d X_d)$ if M contains all non-empty singular subslices of λ, i.e., $M = \{(G_1\{x_1\}, \ldots, G_d\{x_d\}) \mid (G_1\{x_1\}, \ldots, G_d\{x_d\}) \sqsubseteq \lambda\}$.*

For each slice there is a unique decomposition. From Lemma 1 follows that if M is a decomposition of a slice λ then M represents the same recurrence as λ, i.e., $\mathcal{I}(M) = \mathcal{I}(\lambda)$. Lemma 2 states if M is a decomposition of λ then the unions of selectors at the corresponding levels in M yield the selectors of λ.

Lemma 2. *If a singular multislice $M = \{\dot{\lambda}_1, \ldots, \dot{\lambda}_p\}$ is a decomposition of a slice $\lambda = (G_1 X_1, \ldots, G_d X_d)$ and for all $i \in [1, p] : \dot{\lambda}_i = (G_1\{x_{1,i}\}, \ldots, G_d\{x_{d,i}\})$ then for all $l \in [1, d] : X_l = \bigcup_{i=1}^{p} \{x_{l,i}\}$.*

Consider the slice $\lambda_2 = (yea\{7\},\ wee\{1\},\ day\{4\},\ hou\{6\text{-}7\},\ min\{20,40,50\})$. The following multislice M_{λ_2} is a decomposition of λ_2 and represents the same recurrence as λ_2. The unions of selectors at the corresponding levels give the selectors of λ_2.

$$\begin{aligned} M_{\lambda_2} = \{&(yea\{7\}, wee\{1\}, day\{4\}, hou\{6\}, min\{20\}),\\ &(yea\{7\}, wee\{1\}, day\{4\}, hou\{6\}, min\{40\}),\\ &(yea\{7\}, wee\{1\}, day\{4\}, hou\{6\}, min\{50\}),\\ &(yea\{7\}, wee\{1\}, day\{4\}, hou\{7\}, min\{20\}),\\ &(yea\{7\}, wee\{1\}, day\{4\}, hou\{7\}, min\{40\}),\\ &(yea\{7\}, wee\{1\}, day\{4\}, hou\{7\}, min\{50\})\} \end{aligned}$$

Let M be a singular multislice representing a recurrence P with a hierarchy $(G_1, \ldots, G_d)$ that contains no empty slices. Let $M_{\min} = \{\lambda_1, \ldots, \lambda_q\}$ be a minimum multislice representation of P with the same hierarchy. From Lemma 1, each singular subslice $\dot{\lambda}$ of each slice $\lambda_i \in M_{\min}$ is in M, i.e., $\forall \lambda_i \in M_{\min}(\forall \dot{\lambda} \sqsubseteq \lambda_i(\dot{\lambda} \in M))$. The decomposition M_{λ_i} of each slice $\lambda_i \in M_{\min}$ is then a subset of M, $M_{\lambda_i} \subseteq M$.

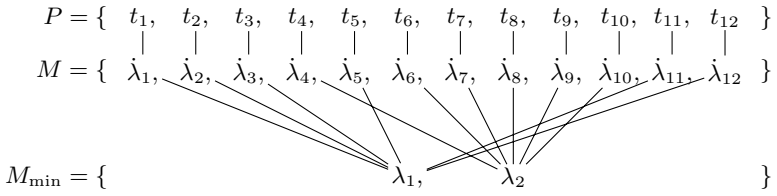


Fig. 3. Relationships between P, M, and $M_{\min}$ from Examples 2 and 4

Example 5. Consider the singular multislice $M = \{\dot{\lambda}_1, \ldots, \dot{\lambda}_{12}\}$ from Example 2. Each singular slice in M represents a distinct time instant in P. A minimal multislice representation $M_{\min}$ of P with hierarchy (*yea*, *wee*, *day*, *hou*, *min*) consists of two slices λ_1 and λ_2:

$$\begin{aligned} M_{\min} = \{&\lambda_1 = (yea\{7\}, wee\{1\}, day\{4\}, hou\{5\text{-}6,8\}, min\{10,30\}),\\ &\lambda_2 = (yea\{7\}, wee\{1\}, day\{4\}, hou\{6\text{-}7\}, \quad min\{20,40,50\})\} \end{aligned}$$

The two corresponding decompositions $\{\dot{\lambda}_1, \dot{\lambda}_2, \dot{\lambda}_3, \dot{\lambda}_5, \dot{\lambda}_{11}, \dot{\lambda}_{12}\}$ and $\{\dot{\lambda}_4, \dot{\lambda}_6, \dot{\lambda}_7, \dot{\lambda}_8, \dot{\lambda}_9, \dot{\lambda}_{10}\}$ are subsets of M and cover all slices in M. Figure 3 illustrates the relationships between P, M, and $M_{\min}$ from Examples 2 and 4.

Theorem 1. *Let P be a finite recurrence and let $(G_1, \ldots, G_d = G_\perp)$, $d > 1$, be a hierarchy of time granularities. Finding a minimum multislice representation of P with the hierarchy $(G_1, \ldots, G_d = G_\perp)$ is an NP-hard problem.*

Proof. The problem of finding a minimal multislice representation of P with the hierarchy $(G_1, \ldots, G_d = G_\perp)$ can be formulated as the following problem Π_1 :

Given a singular multislice M which contains no empty slices. Find the minimal number of subsets of M such that each subset is a decomposition of some slice and all subsets cover all slices in M.

We prove that Π_1 is NP-hard by reducing a known NP-complete problem of covering a bipartite graph by complete bipartite subgraphs (appears as the problem GT18 in [5]) to Π_1. We formulate the problem of covering a bipartite graphs as the problem Π_4 : Given a bipartite graph (A, B, E), where A, B are disjoint sets of vertexes and $E \subseteq \{\{a, b\} \mid a \in A, b \in B\}$ is a set of edges. Given a natural number q, $1 \leq q \leq |E|$. Are there q complete bipartite subgraphs $(A'_1, B'_1, E'_1), \ldots, (A'_q, B'_q, E'_q)$, where $A'_i \subseteq A$, $B'_i \subseteq B$, and $E'_i = \{\{a, b\} \mid a \in A'_i, b \in B'_i\}$, such that they cover all edges in E, i.e., $E = \bigcup_{i=1}^{q} E'_i$?

We define intermediate problems Π_2, Π_3, and then prove that $\Pi_4 \leq_T \Pi_3 \leq_T \Pi_2 \leq_T \Pi_1$, where $\leq_T$ stands for Turing reducible. The problem Π_1 is an optimization problem. Using the fact that the size of a minimal multislice representation is always between 1 and $|M|$, we can formulate the following decision problem Π_2 : Given a singular multislice M which contains no empty slices, and a natural number q, $1 \leq q \leq |M|$. Are there q subsets of M such that each subset is a decomposition of some slice and all subsets cover all slices in M?

Having the solution to Π_1, we can solve the problem Π_2 in constant time. By proving that Π_2 is NP-complete we show that Π_1 is NP-hard. Π_2 is certainly in NP: if we guess q subsets of M we can check if they are decompositions of some slices in polynomial time using Lemma 2. For $d = 2$ the problem Π_2 is formulated as the following problem Π_3 : Given two sets X_1, X_2 and a multislice $M \subseteq \{(G_1\{x_1\}, G_2\{x_2\}) \mid x_1 \in X_1 \wedge x_2 \in X_2\}$. Given a natural number q, $1 \leq q \leq |M|$. Are there q subsets of M such that each subset is a decomposition of some slice and all subsets cover all slices in M?

The problem Π_4 is equivalent to the problem Π_3, where every $a \in A$ corresponds to $x_a \in X_1$, every $b \in B$ corresponds to $x_b \in X_2$. Each edge $\{a, b\} \in E$ corresponds to a slice $(G_1\{x_a\}, G_2\{x_b\}) \in M$. A complete bipartite subgraph in (A, B, E) corresponds to a decomposition in M. Figure 4(a,b) shows an example of such a correspondence. Note, that the bipartite graph in Fig. 4(a) can be covered by its four complete bipartite subgraphs, and even though the subgraph drawn with bold lines is the largest complete bipartite subgraph in the given graph, it does not belong to these four (see Fig. 4(d)).

We can reduce a problem Π_3 with $d = 2$ to a problem Π_2 with $d \geq 2$ just by adding the same pairs granularity-selector to each slice in M. For example, we can map the multislice from Fig. 4(b) to the multislice in Fig. 4(c). □

3.4 Level-Wise Merge Algorithm `LMerge`

Recall that `BMerge` does not impose any ordering in the merging phase. Here we present the algorithm `LMerge` (Level-wise Merge) which imposes a specific order on the merging process: slices are merged by granularities, that is, for each hierarchy level l the algorithm performs all possible merges across l before it begins to merge across another level. Within a hierarchy level the order in which slices are merged is irrelevant due to the associativity of the merge operation.

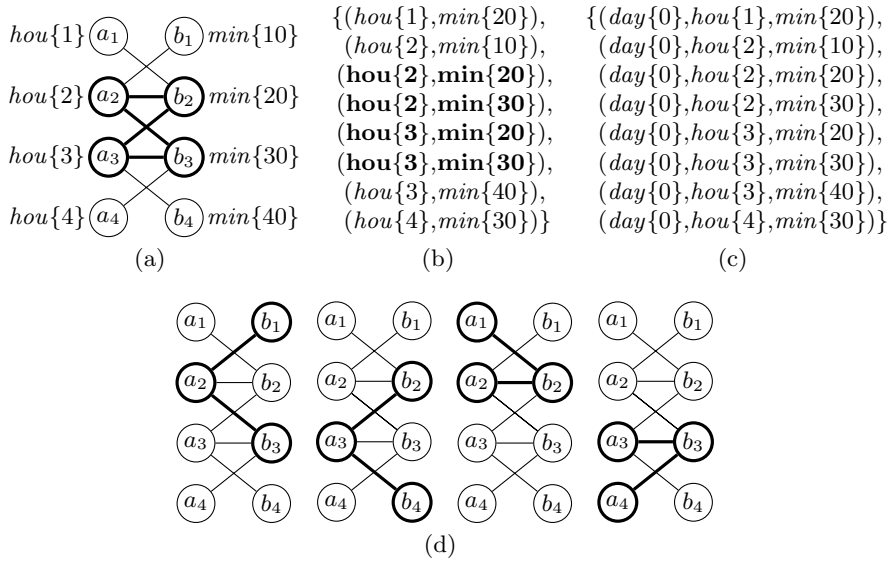


Fig. 4. A bipartite graph with a complete bipartite subgraph (a), the corresponding singular multislice with the corresponding decomposition (b), the corresponding multislice with $d = 3$ (c), and the smallest cover of the bipartite graph by complete bipartite subgraphs (d)

The `LMerge` algorithm adopts the same bottom-up strategy as the baseline algorithm: (Step 1) a singular multislice M with the given hierarchy $(G_1, \ldots, G_d)$ is constructed and (Step 2) mergeable slices in M are merged. The main loop in Step 2 iterates through all levels of the hierarchy $(G_1, \ldots, G_d)$ and determines the order in which slices are merged. At each iteration the slices in M are sorted on all selectors except the one that corresponds to level l of the iteration. For instance, if the hierarchy is (*yea*, *wee*, *day*, *hou*, *min*) and the iteration level is of *day*, the slices are sorted on the selectors of *yea*, *wee*, *hou*, and *min*. Such sorting brings the slices that are mergeable across l together into contiguous clusters such that in a single pass through the multislice each cluster can be merged into a single slice. The functions first and next retrieve the first and next slice from M, respectively.

`LMerge` runs in $O(d^2 \cdot |P| \cdot \log |P|)$ time, where $O(d \cdot |P| \cdot \log |P|)$ is the time complexity of sorting.

Example 6. Consider the singular multislice M from Example 2. The first iteration of the main loop merges across the granularity *min*. The sorting step produces the four clusters $\{\dot{\lambda}_1, \dot{\lambda}_2\}$, $\{\dot{\lambda}_3, \dot{\lambda}_4, \dot{\lambda}_5, \dot{\lambda}_6, \dot{\lambda}_7\}$, $\{\dot{\lambda}_8, \dot{\lambda}_9, \dot{\lambda}_{10}\}$, and $\{\dot{\lambda}_{11}, \dot{\lambda}_{12}\}$, which in a single pass are merged into four slices, yielding

$$
\begin{aligned}
M = \{&(\mathit{yea}\{7\}, \mathit{wee}\{1\}, \mathit{day}\{4\}, \mathit{hou}\{5\}, \mathit{min}\{10{,}30\}),\\
&(\mathit{yea}\{7\}, \mathit{wee}\{1\}, \mathit{day}\{4\}, \mathit{hou}\{6\}, \mathit{min}\{10{,}20{,}30{,}40{,}50\}),\\
&(\mathit{yea}\{7\}, \mathit{wee}\{1\}, \mathit{day}\{4\}, \mathit{hou}\{7\}, \mathit{min}\{20{,}40{,}50\}),\\
&(\mathit{yea}\{7\}, \mathit{wee}\{1\}, \mathit{day}\{4\}, \mathit{hou}\{8\}, \mathit{min}\{10{,}30\})\}.
\end{aligned}
$$

Algorithm `LMerge`

input: recurrence P, hierarchy $(G_1, \ldots, G_d)$
output: multislice M
// Step 1: build a singular multislice
$M := \varnothing$;
for each $t \in P$ **do**
  $M := M \cup \{\text{singular slice for } t\}$;
// Step 2: merge slices level−wise
for $l \in \{1, \ldots, d\}$ **do**
  Sort M on $X_1, \ldots, X_{l-1}, X_{l+1}, \ldots, X_d$;
  $\lambda := \text{first}(M)$;
  while $\lambda \neq \mathit{null}$ **do**
    $\lambda' := \text{next}(M)$;
    while mergeable(λ, λ', l) **do**
      $\lambda := \lambda + \lambda'$;
      $M := M \setminus \{\lambda'\}$;
      $\lambda' := \text{next}(M)$;
    $\lambda := \lambda'$;
return M;

The second iteration merges across the level of *hou*. The slices are sorted on the selectors of the granularities *yea*, *wee*, *day*, and *min*, yielding three clusters, which are merged to get the multislice

$$\begin{aligned} M = \{&(\mathit{yea}\{7\}, \mathit{wee}\{1\}, \mathit{day}\{4\}, \mathit{hou}\{6\}, \quad \mathit{min}\{10,20,30,40,50\}),\\ &(\mathit{yea}\{7\}, \mathit{wee}\{1\}, \mathit{day}\{4\}, \mathit{hou}\{7\}, \quad \mathit{min}\{20,40,50\}),\\ &(\mathit{yea}\{7\}, \mathit{wee}\{1\}, \mathit{day}\{4\}, \mathit{hou}\{5,8\}, \mathit{min}\{10,30\})\} \end{aligned}$$

This multislice is final, and the iteration through the granularities of *day*, *wee*, and *yea* does not provide further merging.

Lemma 3. *Let P be a recurrence and $(G_1, \ldots, G_d)$ be a hierarchy. The algorithm `LMerge` returns a final multislice M.*

Proof. To keep the proof simple, we assume the merging is done from level 1 to level d (the lemma holds for any order of levels). After m–1 iterations the selectors at levels $m, \ldots, d$ of all slices are still unmodified and consist of single integers. We do a proof by contradiction. Assume the result of `LMerge` is not final and contains two slices $\lambda_X = (G_1 X_1, \ldots, G_d X_d)$ and $\lambda_Y = (G_1 Y_1, \ldots, G_d Y_d)$ that are mergeable across some level $m \in [1, d]$. Then, $X_l = Y_l$ for all levels $l \neq m$, and there is a subslice $\lambda'_X = (G_1 X_1, \ldots, G_{m-1} X_{m-1}, G_m\{x_m\}, \ldots, G_d\{x_d\})$ of λ_X and a subslice $\lambda'_Y = (G_1 Y_1, \ldots, G_{m-1} Y_{m-1}, G_m\{y_m\}, \ldots, G_d\{y_d\})$ of λ_Y such that $x_m \neq y_m$ and $x_l = y_l$ for all $l \in [m+1, d]$. Such a situation is impossible, because slices λ'_X and λ'_Y should already have been merged after iteration m, which leads to a contradiction. □

4 Analytical Evaluation

Let P be a finite recurrence, $M_{\min}$ be a minimal multislice representation of P with a hierarchy $(G_1, \ldots, G_d)$, and M be a multislice representation of P with the same hierarchy constructed by `LMerge`. We define the worst case of `LMerge` as maximum difference $|M| - |M_{\min}|$. In the following we show that for $d = 2$ in the worst case $|M| < \min\left\{2^{|M_{\min}|}, \frac{2|P|}{|M_{\min}|}\right\}$. This means that in the worst case the multislice produced by `LMerge` can be exponentially larger than a minimum multislice representation of P, however, still less than P by the factor of $\frac{|M_{\min}|}{2}$. To give an intuition, for $d = 2$ and $|M_{\min}| = 10$ in the worst case $|M| = 1023$ and $|P|$ is at least 5120. In Section 5 we show that real recurrences from bus schedules are far from this worst case, and the `LMerge` algorithm provides an average compression ratio of 99%.

To show the rationale of these bounds we introduce a geometric visualization of singular multislices. A singular multislice with a hierarchy (G_1, G_2) can be visualized as a set of points in 2D space. Consider the singular multislice in Fig. 5(a). This multislice has hierarchy (*hou*, *min*) and can be visualized in 2D space where one dimension corresponds to the granularity *hou* and the other dimension corresponds to the granularity *min* (see Fig. 5(b)). This geometric visualization allows to observe two properties. First, all slices laying on the same line parallel to the *min* axis are mergeable across the granularity *min*, and all slices laying on the same line parallel to the *hou* axis are mergeable across the granularity *hou*. Second, a subset of slices which fills a rectangle in the geometric visualization is the decomposition of some slice. For example, the subset of slices connected with dotted lines in Fig. 5(b) forms the decomposition of the slice (*hou*{6-7}, *min*{40,50}).

Note, that both segments and points are special cases of a rectangle and visualize the decompositions of some slices. A change of order of values on both axes does not change any of the two properties. This means that a subset of points visualizes a decomposition if there are two permutations of values on both axes for which it fills a rectangle. Figure 5(c) visualizes a multislice with all slices grouped into two decompositions marked with white and black circles. The corresponding slices make up a minimal multislice representation of the recurrence represented by the singular multislice in Fig. 5(a).

Applying `LMerge` to the multislice in Fig. 5(a) the slices are merged iteratively across the two hierarchy levels. For example, in Fig. 5(d) the slices that are merged in the first iteration across the level of *min* are connected into segments (dotted lines). In the second iteration, the mergeable slices are merged across the level of *hou*. In our geometric visualization the mergeable slices would be the segments that fill a rectangle for some permutation of indexes of *hou*. For example, in Fig. 5(d) there are three groups of such segments making up three decompositions that are marked with white circles, white squares, and black circles. For comparison, Fig. 5(e) visualizes 4 decompositions that are found by `BMerge` when the singular slices are chronologically ordered.

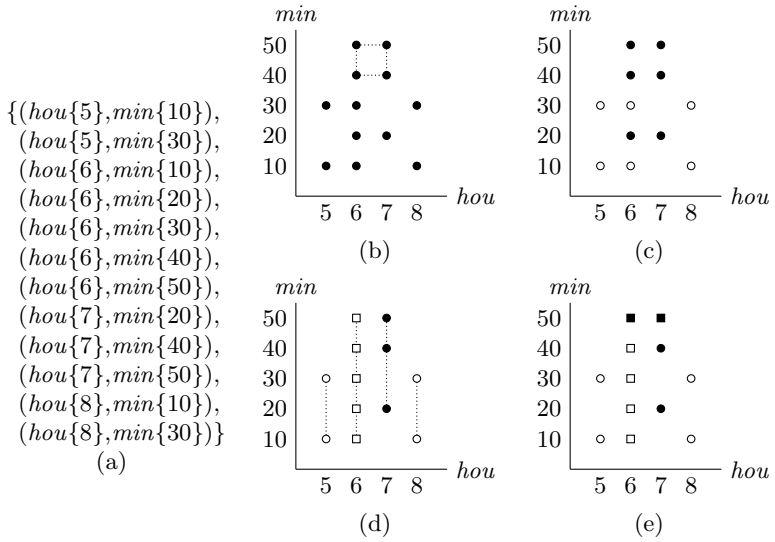


Fig. 5. A singular multislice with $d = 2$ (a), its geometric visualization in 2D space (b), grouping into minimal number of decompositions (c), grouping into decompositions by `LMerge` with merging across levels 1,2 (d), grouping into decomposition by `BMerge` (chronological order) (e)

Figure 6(a) shows the worst case for `LMerge` when merging is done first across G_1 and then across G_2. When merging first across G_2, the singular slices visualized in this figure can be merged into 4 slices and it is a minimal multislice representation of the corresponding recurrence. Merging first across G_1 would return 15 slices which corresponds to the number of all possible intersections of 4 sets. We can systematically construct recurrences P for which `LMerge`$(P, (G_1, G_2))$ returns $2^{|M_{\min}|} - 1$ slices. We can construct the cases where merging first across G_1 or G_2 does not avoid the exponential difference. For example, Fig. 6(b) visualizes a singular multislice which can be compressed into 8 slices, however, `LMerge` algorithm would return 19. In such cases `LMerge` would return at least $2^{\frac{|M_{\min}|}{2}} - 1 + |M_{\min}|$ slices.

The singular multislice visualized in Fig. 6(a) contains 32 singular time slices. In order to construct the worst case where $|M| = 2^{|M_{\min}|} - 1$, we need at least $|P| = |M_{\min}| \cdot 2^{|M_{\min}|-1}$ singular slices. From here, $|M| = \frac{2|P|}{|M_{\min}|} - 1$.

5 Empirical Evaluation

For the empirical evaluation we implemented `BMerge` and `LMerge` in PostgreSQL. We used the `BMerge` algorithm with two different orderings of the singular multislices: when the singular slices are ordered chronologically (`BMerge`, chronological order), and when the singular slices are ordered randomly (`BMerge`, random

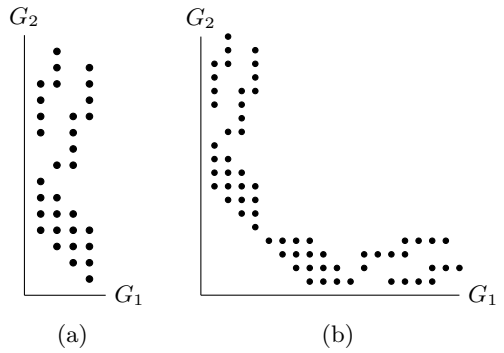


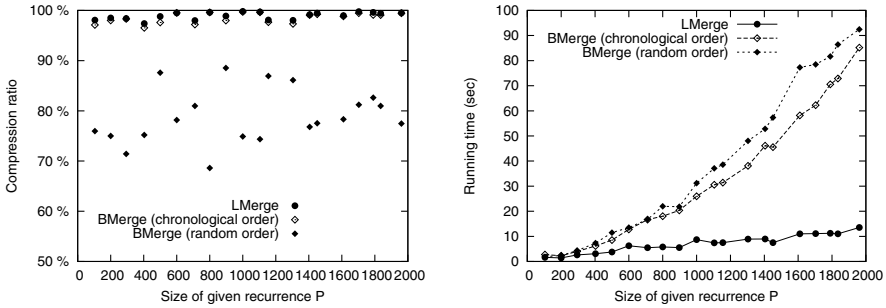
Fig. 6. The worst case for `LMerge` with merging across levels $1, 2$ (a), and a bad case for `LMerge` for any order of levels (b)

order). The randomization is achieved by reordering the multislice according to a sequence of randomly generated numbers using PostgreSQL system functions. In all experiments we used the hierarchy (wee, day, hou, min)[2]. For the `LMerge` algorithm we tried all 24 possible orders in which `LMerge` can iterate through the levels of the hierarchy (wee, day, hou, min). In the plots below for the `LMerge` algorithm we show the average compression ratio and the average running time over all 24 possible orders.

In the first experiment we compare `LMerge` and `BMerge` on the real-world data from the bus network of Bozen-Bolzano from the first half of the year 2007. This data describes 384 different route options grouped into 18 main routes. We selected 20 route options with recurrences of departures with sizes uniformly distributed within the range $[100, 2000]$. The best theoretically possible average compression for these recurrences is 99.64% assuming that each of the recurrences can be represented with a multislice of size 2. Figure 7 shows the results of this experiment. Algorithm `LMerge` provides with an average compression ratio of 98.93% which is very close to the optimal solution (i.e., minimal multislice). Changing the order of levels in the `LMerge` algorithm does not significantly impact the compression ratio for the selected recurrences. The average compression for the best orders is 99.01% and for the worst orders 98.81%. Algorithm `BMerge` with the random order is visibly behind `LMerge` providing on average 78.94% compression. Algorithm `BMerge` with the chronological order is very close to `LMerge` providing an average compression ratio of 98.52%. The running time of both algorithms conforms to our asymptotic bounds: $O(d^2 \cdot |P| \cdot \log |P|)$ for the `LMerge` algorithm and $O(d \cdot |P|^2)$ for the `BMerge` algorithm.

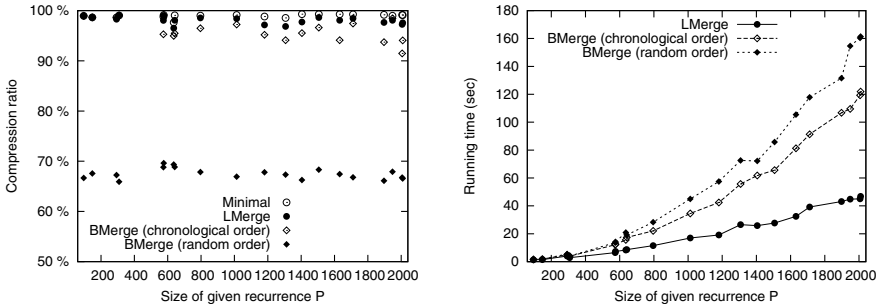
For the second experiment we generated 20 multislices representing recurrences from the first half of the year 2007 with the size varying within the range $[96, 2009]$. The sizes of the generated multislices cover the range $[1, 20]$ and are

[2] Since all recurrences are within the year 2007, the granularity *yea* provides no additional compression, hence we omitted it.



(a) Comparison of compression ratios. (b) Comparison of running time.

Fig. 7. Comparison of multislices produced by `LMerge` and `BMerge` for 20 real-world recurrences



(a) Comparison of compression ratios. (b) Comparison of running time.

Fig. 8. Comparison of multislices produced by `LMerge` and `BMerge` for 20 generated recurrences

taken for known minimal representations providing the average compression ratio of 98.94%. Figure 8 shows the results of this experiment. Algorithm `LMerge` provides with an average compression ratio of 98.04%. Changing the order of levels in `LMerge` has a bigger impact than in the previous experiment (more than 1% in compression). The average compression for the best orders is 98.62% and for the worst orders 97.19%. In this experiment the difference between `LMerge` and `BMerge` has increased for both versions of the `BMerge` algorithm. `BMerge` with the random order provides on average 67.50% compression. `BMerge` with the chronological order provides an average compression ratio of 96.23%.

6 Related Work

Multislices are based on various formalisms coming from the research community [4,6,7,2] and generalize some known representations used in industry [8,9].

Time slices originate from the works of Leban et al. [6] and Niezette et al. [4]. Niezette et al. coined the term time slice and defined the intersection operation on time slices. In their work the authors used sets of slices to overcome the limitation of expressiveness of single time slices. Kasperovics et al. [1] introduced multislices as a basic object for representing recurrences, defined the difference operation on time slices and multislices, and proposed a scalable representation of multislices in relational databases. There is a number of works improving the expressiveness of time slices (e.g., [10,11,12,2]), or incorporating them into more complex representation formalisms (e.g., [13,7,14]). In this paper we presented an operation that constructs multislice representation for a given recurrence, which was not addressed by the previous works.

There are few works that address constructing compact representations from given recurrences. These representations, however, favor periodic recurrences and do not use time granularities. Behr et al. [15] proposed a compact representation formalism, called periodic moving tree, for periodic moving objects and provided with an algorithm for constructing periodic moving trees from recurrences. Work by Bettini et al. [16] proposed an algorithm for minimizing the representations of periodic sets which can be used for constructing compact representations of periodic recurrences. Multislices provide a high compression for recurrences aligned to the hierarchies of time granularities. Such recurrences are common for many kinds of schedules and are less periodic because of monthly or yearly repetitions, and because of multiple exceptions, such as public holidays. The representation of such recurrences with periodic moving trees or periodic sets would require more space.

7 Conclusions and Future Work

In this paper we studied the problem of constructing multislice representations for non-empty finite recurrences. We proved that the construction of a minimal multislice representation is an NP-hard problem and proposed a scalable approximation algorithm `LMerge`. Although in the worst case `LMerge` might produce an exponentially worse compression than the minimal solution, experiments with real-world data show that the multislices computed by the algorithm provide a very high compression ratio of 99%, which is very close to the optimal solution. `LMerge` clearly outperforms a straightforward baseline algorithm `BMerge` both in terms of compression and in terms of time.

The recurrences in public transport schedules, lecture schedules, and personal calendars are often a subject of changes, and so would be their multislice representations. The changes can be resolved using the union and difference operations on multislices [1], which in most would decrease the compression. The ideas presented in this paper can be extended for compressing multislices in more general settings, where multislices are not necessarily singular (e.g., when produced as the result of operations on multislices).

References

1. Kasperovics, R., Böhlen, M.H., Gamper, J.: Evaluating exceptions on time slices. In: Laender, A.H.F. (ed.) ER 2009. LNCS, vol. 5829, pp. 251–264. Springer, Heidelberg (2009)
2. Ohlbach, H.J.: Periodic temporal notions as 'tree partitionings'. Forschungsbericht/Research Report PMS-FB-2006-11, Institute for Informatics, University of Munich (2006)
3. Dawson, F., Stenerson, D.: Internet calendaring and scheduling core object specification, iCalendar (1998)
4. Niezette, M., Stévenne, J.M.: An efficient symbolic representation of periodic time. In: Proceedings of the First International Conference on Information and Knowledge Management, pp. 161–168
5. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
6. Leban, B., McDonald, D.D., Forster, D.R.: A representation for collections of temporal intervals. In: Proceedings of AAAI 1986, August 1986, pp. 367–371 (1986)
7. Terenziani, P.: Symbolic user-defined periodicity in temporal relational databases. IEEE Trans. Knowl. Data Eng. 15(2), 489–509 (2003)
8. Google Inc.: Google Transit Feed Specification (February 2008)
9. Weber, C., Brauer, D., Kolmorgen, V., Hirschel, M., Provezza, S., Hulsch, T.: Fahrplanbearbeitungssystem FBS – Anleitung. iRFP (September 2006)
10. Cukierman, D.R., Delgrande, J.P.: The SOL theory: A formalization of structured temporal objects and repetition. In: Proceedings of TIME, pp. 28–35 (2004)
11. Anselma, L.: Recursive representation of periodicity and temporal reasoning. In: Proceedings of TIME, pp. 52–59 (2004)
12. Kasperovics, R., Böhlen, M.H.: Querying multi-granular compact representations. In: Li Lee, M., Tan, K.-L., Wuwongse, V. (eds.) DASFAA 2006. LNCS, vol. 3882, pp. 111–124. Springer, Heidelberg (2006)
13. Chandra, R., Segev, A., Stonebraker, M.: Implementing calendars and temporal rules in next generation databases. In: Proceedings of ICDE 1994, Washington, DC, USA, pp. 264–273. IEEE Computer Society, Los Alamitos (1994)
14. Ning, P., Wang, X.S., Jajodia, S.: An algebraic representation of calendars. Ann. Math. Artif. Intell. 36(1-2), 5–38 (2002)
15. Behr, T., de Almeida, V.T., Güting, R.H.: Representation of periodic moving objects in databases. In: Proceedings of the 14th International Workshop on Geographic Information Systems (ACM-GIS), pp. 43–50 (2006)
16. Bettini, C., Mascetti, S.: An efficient algorithm for minimizing time granularity periodical representations. In: Proceedings of TIME, pp. 20–25 (2005)